A Centaur tutorial

Ian Jacobs, Laurence Rideau-Gallot

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Sophia Antipolis
B.P. 109
06561 Valbonne Cedex
France
Tél.: 93 65 77 77

Rapports Techniques

N°141

Programme 2
Calcul symbolique, Programmation et Génie logiciel

A Centaur Tutorial

Ian JACOBS
Laurence RIDEAU

Août 1992

A Centaur Tutorial

Ian Jacobs
INRIA Rocquencourt

Laurence Rideau-Gallot
INRIA Sophia-Antipolis

August 18, 1992

A Centaur Tutorial

Abstract

This paper presents the Centaur system through a tutorial describing the creation of an environment for a small language of mathematical expressions called Exp. With Centaur, the user may interactively generate programming language environments, including structured editors, debuggers, interpreters, and other tools. In this tutorial, all phases of language specification are covered: the design of the abstract and concrete syntax of Exp in Metal and Sdf, the pretty printing rules in Ppml, and the semantics of an Exp interpreter in Typol. The tools generated by Centaur based on these specifications are enhanced by a user interface built with Centaur graphic primitives.

Une Introduction au système Centaur

Résumé

Dans ce papier, nous preśentons le système Centaur qui est un outil interactif de génération d'environnements de langages de programmation, c'est à dire des éditeurs structurés, des interprètes, et d'autres outils de mise au point. Nous décrivons la création d'un environnement pour un petit langage d'expressions mathématiques appelé Exp. Toutes les étapes de spécification du langage sont parcourues: la spécification de la syntaxe abstraite et de la syntaxe concrète de Expen Metal et en Sdf, les règles d'affichage en Ppml, et la sémantique d'un évaluateur d'expressions en Typol. De plus, les outils générés sont intégrés dans une interface homme-machine construite à partir des primitives graphiques de Centaur.

Contents

1 Introduction

In this tutorial, we walk through the development of an environment for a language of mathematical expressions called Exp. We highlight important notions along the way and explain all the steps necessary for creating tools for Exp.

1.1 The Exp environment

1.1.1 Syntax and parsers

We specify the syntax of Exp with both the Sdf and Metal formalisms, which we then compile to produce an abstract syntax (formalism) definition, a parser, and tree building functions which the parser uses to construct Vtp abstract syntax trees from syntactically correct programs. All parsers generated from Metal specifications are external processes that communicate with Centaur.

1.1.2 Pretty printers

We specify the pretty printer with Ppml, which we compile to produce an incremental tree formatter. We may create a compiled Le-Lisp module for this pretty printer specification to improve performance.

1.1.3 Evaluator

We specify an evaluator for Exp in Typol, which is compiled into a Prolog program. The evaluator reads Exp abstract trees, and according to the semantics we prescribe in Typol, evaluates it, returning an abstract syntax tree that represents the result.

1.1.4 Graphic interface

We construct a graphic interface that allows us to trigger the evaluator with the mouse and display the results in a special window.

1.2 Resources

Centaur now features a *resource manager* by which users provide values for most system parameters, such as formalism locations, pretty printer colors and fonts, and ctedit mouse event lists. The resource manager consults the *current database*, which contains a list of resource specifications. Each specification assigns a value to an object parameter, called an object *property*.

A resource specification is a pair that assigns a value to a pattern. Each pattern represents the path leading to an object in the system hierarchy. The last word in the pattern is a property

of the object for which we may furnish a value, such as the background color used by a pretty printer or the location of a formalism. Patterns are modeled after those used by the X window system.[1]

We specify resources in *database files* that CENTAUR reads and stores in the current database. At the beginning of a session, CENTAUR automatically loads the file named `$HOME/.centaur.rdb` into the current database. This file contains a specification that "declares" new user languages, and possible other specifications that direct CENTAUR to separate database files. We organize database files hierarchically, mirroring the organization of system objects. Each database file contains resource information pertinent to a given object, and then directs us to database files that concern subobjects.

Except for the `.centaur.rdb` file, database files are loaded on demand only. Each time we modify a database file, however, we reread specifications into the current database by clicking on the button *Reset Resources*. This alone does not suffice to update our working environment, however, since tools in the environment only consult the database when they require parameter values. Thus, after resetting the database, we must also reset pertinent tools by hand so that they read fresh database values.

If at any time you wish to print the contents of the current database, type the line `(print-database)` in the CENTAUR main window. We discuss resource specifications and construct databases throughout the tutorial.

1.3 The Centaur environment

Aside from the specification languages, CENTAUR provides several generic tools that facilitate the development of an environment prototype.

1.3.1 The ctedit

The *ctedit* is a structured editor which ensures syntactic correctness when you edit a program written in a language `L`. The ctedit allows only tree surgery that obeys a language's abstract syntax definition.

1.3.2 The ctview

The *ctview* is a viewing tool which allows us to manipulate programs through a graphic interface. Clicking on the *Editor* button of the CENTAUR main menu opens a new ctview which contains an empty ctedit. Each ctview menu bar has the following pulldown menus:

[1] Long resource specifications may be split on to several lines. Each line (except the last) should end with a backslash follwed only by a newline. Due to a bug in X11R4 (corrected in X11R5), continued lines must not begin with white space.

- *File* : This menu allows you to read and parse programs from external files into the ctedit, and write programs.
- *Display* : This menu allows you to modify certain pretty printing parameters (e.g., print level, page width), change pretty printers, and refresh the ctedit contents.
- *Edit* : Allows you to perform generic edition: cut, copy, paste, or send selected subexpressions to a separate Gnu emacs editor for unstructured edition.
- *Selections* : Once you have read a program into a ctview, you may select program fragments by clicking and dragging with the mouse. This menu allows you to fine tune the presentation of selected material. You may make a selection visible, shrink it (set its local pretty print level to zero), expand it (the opposite), or unselect it.

When we read a program written in `L` into a ctview, it becomes specific to the language `L`. The ctedit obeys `L`'s syntax definition. If we have described a graphical environment for `L`, the ctview is transformed by the addition of the environment's buttons, menus, mouse behavior, etc.

1.3.3 Error handling

Language environments may use a generic mechanism for displaying error messages. For example, errors incurred when type checking a METAL, PPML, or TYPOL program appear in a special window. Clicking on an error message and then the *Show Error* button selects and highlights the error in the ctview containing the program.

Syntax errors encountered while parsing programs appear in the CENTAUR window. Syntax errors prevent the construction of an abstract syntax tree, so no tree appears in the editor.

1.3.4 Parsers

CENTAUR starts external parsers for any known language on demand, according to the system resources. These parsers continue to run even when you exit CENTAUR so that you may use them for another session.

1.4 Preparation

Before attacking the design of the EXP environment, we must make a few preparations.

1.4.1 The Exp directory

Although you may create a language directory anywhere you like, we suggest creating a directory: